|  |  |  |
| --- | --- | --- |
| Java Annotations | | |
|  |  |  |
|  | | |

# Java Annotations

Java **Annotation** is a tag that represents the *metadata* i.e. attached with class, interface, methods or fields to indicate some additional information which can be used by java compiler and JVM.

Annotations in Java are used to provide additional information, so it is an alternative option for XML and Java marker interfaces.

First, we will learn some built-in annotations then we will move on creating and using custom annotations.

## Built-In Java Annotations

There are several built-in annotations in Java. Some annotations are applied to Java code and some to other annotations.

## Built-In Java Annotations used in Java code

* @Override
* @SuppressWarnings
* @Deprecated

## Built-In Java Annotations used in other annotations

* @Target
* @Retention
* @Inherited
* @Documented

## Understanding Built-In Annotations

Let's understand the built-in annotations first.

## @Override

@Override annotation assures that the subclass method is overriding the parent class method. If it is not so, compile time error occurs.

Sometimes, we does the silly mistake such as spelling mistakes etc. So, it is better to mark @Override annotation that provides assurity that method is overridden.

1. **class** Animal{
2. **void** eatSomething(){System.out.println("eating something");}
3. }
5. **class** Dog **extends** Animal{
6. @Override
7. **void** eatsomething(){System.out.println("eating foods");}//should be eatSomething
8. }
10. **class** TestAnnotation1{
11. **public** **static** **void** main(String args[]){
12. Animal a=**new** Dog();
13. a.eatSomething();
14. }}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestAnnotation1)

Output:Comple Time Error

## @SuppressWarnings

@SuppressWarnings annotation: is used to suppress warnings issued by the compiler.

1. **import** java.util.\*;
2. **class** TestAnnotation2{
3. @SuppressWarnings("unchecked")
4. **public** **static** **void** main(String args[]){
5. ArrayList list=**new** ArrayList();
6. list.add("sonoo");
7. list.add("vimal");
8. list.add("ratan");
10. **for**(Object obj:list)
11. System.out.println(obj);
13. }}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestAnnotation2)

Now no warning at compile time.

If you remove the @SuppressWarnings("unchecked") annotation, it will show warning at compile time because we are using non-generic collection.

## @Deprecated

@Deprecated annoation marks that this method is deprecated so compiler prints warning. It informs user that it may be removed in the future versions. So, it is better not to use such methods.

1. **class** A{
2. **void** m(){System.out.println("hello m");}
4. @Deprecated
5. **void** n(){System.out.println("hello n");}
6. }
8. **class** TestAnnotation3{
9. **public** **static** **void** main(String args[]){
11. A a=**new** A();
12. a.n();
13. }}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestAnnotation3)

## At Compile Time:

Note: Test.java uses or overrides a deprecated API.

Note: Recompile with -Xlint:deprecation for details.

## At Runtime:

hello n

## Java Custom Annotations

**Java Custom annotations** or Java User-defined annotations are easy to create and use. The *@interface* element is used to declare an annotation. For example:

1. **@interface** MyAnnotation{}

Here, MyAnnotation is the custom annotation name.

## Points to remember for java custom annotation signature

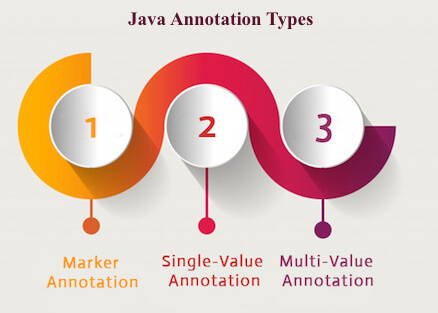
There are few points that should be remembered by the programmer.

1. Method should not have any throws clauses
2. Method should return one of the following: primitive data types, String, Class, enum or array of these data types.
3. Method should not have any parameter.
4. We should attach @ just before interface keyword to define annotation.
5. It may assign a default value to the method.

## Types of Annotation

There are three types of annotations.

1. Marker Annotation
2. Single-Value Annotation
3. Multi-Value Annotation



## 1) Marker Annotation

An annotation that has no method, is called marker annotation. For example:

1. **@interface** MyAnnotation{}

The @Override and @Deprecated are marker annotations.

## 2) Single-Value Annotation

An annotation that has one method, is called single-value annotation. For example:

1. **@interface** MyAnnotation{
2. **int** value();
3. }

We can provide the default value also. For example:

1. **@interface** MyAnnotation{
2. **int** value() **default** 0;
3. }

## How to apply Single-Value Annotation

Let's see the code to apply the single value annotation.

1. @MyAnnotation(value=10)

The value can be anything.

## 3) Multi-Value Annotation

An annotation that has more than one method, is called Multi-Value annotation. For example:

1. **@interface** MyAnnotation{
2. **int** value1();
3. String value2();
4. String value3();
5. }
6. }

We can provide the default value also. For example:

1. **@interface** MyAnnotation{
2. **int** value1() **default** 1;
3. String value2() **default** "";
4. String value3() **default** "xyz";
5. }

## How to apply Multi-Value Annotation

Let's see the code to apply the multi-value annotation.

1. @MyAnnotation(value1=10,value2="Arun Kumar",value3="Ghaziabad")

## Built-in Annotations used in custom annotations in java

* @Target
* @Retention
* @Inherited
* @Documented

## @Target

**@Target** tag is used to specify at which type, the annotation is used.

The java.lang.annotation.**ElementType** enum declares many constants to specify the type of element where annotation is to be applied such as TYPE, METHOD, FIELD etc. Let's see the constants of ElementType enum:

|  |  |
| --- | --- |
| **Element Types** | **Where the annotation can be applied** |
| TYPE | class, interface or enumeration |
| FIELD | fields |
| METHOD | methods |
| CONSTRUCTOR | constructors |
| LOCAL\_VARIABLE | local variables |
| ANNOTATION\_TYPE | annotation type |
| PARAMETER | parameter |

## Example to specify annoation for a class

1. @Target(ElementType.TYPE)
2. **@interface** MyAnnotation{
3. **int** value1();
4. String value2();
5. }

## Example to specify annotation for a class, methods or fields

1. @Target({ElementType.TYPE, ElementType.FIELD, ElementType.METHOD})
2. **@interface** MyAnnotation{
3. **int** value1();
4. String value2();
5. }

## @Retention

**@Retention** annotation is used to specify to what level annotation will be available.

|  |  |
| --- | --- |
| **RetentionPolicy** | **Availability** |
| RetentionPolicy.SOURCE | refers to the source code, discarded during compilation. It will not be available in the compiled class. |
| RetentionPolicy.CLASS | refers to the .class file, available to java compiler but not to JVM . It is included in the class file. |
| RetentionPolicy.RUNTIME | refers to the runtime, available to java compiler and JVM . |

## Example to specify the RetentionPolicy

1. @Retention(RetentionPolicy.RUNTIME)
2. @Target(ElementType.TYPE)
3. **@interface** MyAnnotation{
4. **int** value1();
5. String value2();
6. }

## Example of custom annotation: creating, applying and accessing annotation

Let's see the simple example of creating, applying and accessing annotation.

*File: Test.java*

1. //Creating annotation
2. **import** java.lang.annotation.\*;
3. **import** java.lang.reflect.\*;
5. @Retention(RetentionPolicy.RUNTIME)
6. @Target(ElementType.METHOD)
7. **@interface** MyAnnotation{
8. **int** value();
9. }
11. //Applying annotation
12. **class** Hello{
13. @MyAnnotation(value=10)
14. **public** **void** sayHello(){System.out.println("hello annotation");}
15. }
17. //Accessing annotation
18. **class** TestCustomAnnotation1{
19. **public** **static** **void** main(String args[])**throws** Exception{
21. Hello h=**new** Hello();
22. Method m=h.getClass().getMethod("sayHello");
24. MyAnnotation manno=m.getAnnotation(MyAnnotation.**class**);
25. System.out.println("value is: "+manno.value());
26. }}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestCustomAnnotation1)

Output:value is: 10

[download this example](https://static.javatpoint.com/src/newjdk/annotation.zip)

## How built-in annotaions are used in real scenario?

In real scenario, java programmer only need to apply annotation. He/She doesn't need to create and access annotation. Creating and Accessing annotation is performed by the implementation provider. On behalf of the annotation, java compiler or JVM performs some additional operations.

## @Inherited

By default, annotations are not inherited to subclasses. The @Inherited annotation marks the annotation to be inherited to subclasses.

1. @Inherited
2. **@interface** ForEveryone { }//Now it will be available to subclass also
4. **@interface** ForEveryone { }
5. **class** Superclass{}
7. **class** Subclass **extends** Superclass{}

## @Documented

The @Documented Marks the annotation for inclusion in the documentation.